EAST Search History

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L3	2343	717/124-129.ccls.	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/02 17:06
L4	3368	711/147-152.ccls.	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/02 17:07
L5	1504	711/100.ccls.	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/02 17:07
L6	1186	711/105.ccls.	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/02 17:07
L7	2983	714/5-6.ccls.	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/02 17:07
L8	0	(L3-L7)and ((allocat$4 adj1 (resource data program)) with (memory adj1 (leak$3 carsh fail$4))with detect$4)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/02 17:11
L9	0	(L3-L7)and ((allocat$4 adj1 (resource data program)) with (memory adj1 (leak$3 carsh fail$4)))	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/02 17:12
L10	0	(L3-L7)and(memory adj1 (leak$3 carsh fail$4))	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/02 17:13
L11	65	L3 and(memory adj1 (leak$3 carsh fail$4))	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/02 17:13
L12	46	L4 and(memory adj1 (leak$3 carsh fail$4))	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/02 17:13
L13	19	L5 and(memory adj1 (leak$3 carsh fail$4))	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/02 17:13

EAST Search History

L14	9	L6 and(memory adj1 (leak$3 carsh fail$4))	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/02 17:13
L15	109	L7 and(memory adj1 (leak$3 carsh fail$4))	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/02 17:13
L16	0	L7 and(memory adj1 (leak$3 carsh fail$4))with (allocat$4 adj1 (resource data program))	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/02 17:14
S1	50	("5758347" "20030088604" "4797877" "5675790" "5784698" "4803622" "5315708" "5386536" "5504874" "5664151" "5815731" "5905889" "5964840" "5996013" "6070202" "6094685" "6105053" "6151688" "6233630" "6275857" "6338112" "6353898" "6363410" "6442663" "6484173" "6496864" "6629114" "6694419" "6738886" "6742099" "6742100" "6745312" "6968441" "6968557" "6976021" "7111307" "20020198886" "20030018689" "20050005018" "20050114843" "20060034167" "20060136761" "20060206894" "20060248542" "4493049" "5428823" "5802590" "6170020" "6216192" "6223230").pn.	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/02/28 16:21
S2	1	10/723979	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/02/27 16:57
S3	1	S1 and (time adj1 stamp$3)and (memory adj1 leak$3)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/02/27 17:08
S4	3	S1 and (memory adj1 leak$3)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/02/27 17:10
S5	2	S1 and (time adj1 stamp$3)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/02/27 17:09
S6	1	(memory adj1 leak$3) and (time adj1 stamp$3) with (allocat$4 adj5 resource)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/02/27 17:12

EAST Search History

S7	6	(time adj1 stamp$3) with (allocat$4 adj5 resource)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/02/27 17:14
S8	3	"5590329".pn. "6085029".pn. "6167"."535".pn. "6370684".pn. 2002/010640	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/02/27 17:16
S9	5	"5590329".pn. "6085029".pn. "6167535".pn. "6370684".pn. 2002/0120640	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/02/27 17:17
S10	37	(detect$4 with (memory adj3 leak$3)) and (resource with allocat$4)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/02/27 18:14
S11	479	time with stamp$3 with resource	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/02/28 14:35
S12	13	time with stamp$3 with (allocat$4 adj10 resource)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/02/28 14:36
S13	50	("5758347" "20030088604" "4797877" "5675790" "5784698" "4803622" "5315708" "5386536" "5504874" "5664151" "5815731" "5905889" "5964840" "5996013" "6070202" "6094685" "6105053" "6151688" "6233630" "6275857" "6338112" "6353898" "6363410" "6442663" "6484173" "6496864" "6629114" "6694419" "6738886" "6742099" "6742100" "6745312" "6968441" "6968557" "6976021" "7111307" "20020198886" "20030018689" "20050005018" "20050114843" "20060034167" "20060136761" "20060206894" "20060248542" "4493049" "5428823" "5802590" "6170020" "6216192" "6223230").pn.	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/02/28 16:21
S14	3	S13 and (memory adj5 leak$4)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/02/28 16:22
S15	38	(memory adj10 leak$4)with (allocat$5 adj10 resource)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/01 19:41

EAST Search History

S16	21	(detect$4 adj10 memory adj10 leak$4) and (allocat$5 adj10 resource)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/02/28 16:34
S17	1	10/737404	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/02/28 17:11
S18	50	("5758347" "20030088604" "4797877" "5675790" "5784698" "4803622" "5315708" "5386536" "5504874" "5664151" "5815731" "5905889" "5964840" "5996013" "6070202" "6094685" "6105053" "6151688" "6233630" "6275857" "6338112" "6353898" "6363410" "6442663" "6484173" "6496864" "6629114" "6694419" "6738886" "6742099" "6742100" "6745312" "6968441" "6968557" "6976021" "7111307" "20020198886" "20030018689" "20050005018" "20050114843" "20060034167" "20060136761" "20060206894" "20060248542" "4493049" "5428823" "5802590" "6170020" "6216192" "6223230").pn.	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/01 17:35
S19	4	S18 and (call adj1 stack)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/01 17:47
S20	195	(call adj1 stack)with (identif$8 ID (time adj1 stamp$4))	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/01 17:49
S21	7	(call adj1 stack)with (identif$8 ID (time adj1 stamp$4))and (detect$4 with(memory adj1 leak))	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/02 12:55
S22	0	2004/0133895	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/01 19:40
S23	0	"2004/0133895 A1"	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/01 19:40
S24	1	10/737404	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/01 19:41

EAST Search History

S25	79	(detect$4 with(memory adj1 leak)) with (error exception)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2007/03/02 12:56

Terms used **detecting memory leaking** Found **24,309** of **198,146**

Sort results by: relevance

Display results: expanded form

Save results to a Binder

Search Tips

Open results in a new window

Try an Advanced Search

Try this search in The ACM Guide

Results 1 - 20 of 200 Result page: **1** 2 3 4 5 6 7 8 9 10 next

Best 200 shown Relevance scale

1 Low-overhead memory leak detection using adaptive statistical profiling

Matthias Hauswirth, Trishul M. Chilimbi

October 2004 **ACM SIGPLAN Notices , ACM SIGOPS Operating Systems Review , ACM SIGARCH Computer Architecture News , Proceedings of the 11th international conference on Architectural support for programming languages and operating systems ASPLOS-XI**, Volume 39 , 38 , 32 Issue 11 , 5 , 5

Publisher: ACM Press

Full text available: pdf(159.07 KB) Additional Information: full citation, abstract, references, citings, index terms

Sampling has been successfully used to identify performance optimization opportunities. We would like to apply similar techniques to check program correctness. Unfortunately, sampling provides poor coverage of infrequently executed code, where bugs often lurk. We describe an adaptive profiling scheme that addresses this by sampling executions of code segments at a rate inversely proportional to their execution frequency. To validate our ideas, we have implemented SWAT, a novel memory leak detect ...

Keywords: low-overhead monitoring, memory leaks, runtime analysis

2 Symbolic pointer analysis for detecting memory leaks

Berhard Scholz, Johann Blieberger, Thomas Fahringer

November 1999 **ACM SIGPLAN Notices , Proceedings of the 2000 ACM SIGPLAN workshop on Partial evaluation and semantics-based program manipulation PEPM '00**, Volume 34 Issue 11

Publisher: ACM Press

Full text available: pdf(1.93 MB) Additional Information: full citation, abstract, references, citings, index terms

It is well accepted that pointers are a common source of memory anomalies such as loosing references to dynamic records without deallocating them (also known as memory leaks). This paper presents a novel pointer analysis framework that detects memory leaks by statically analyzing the behavior of programs.

Our approach is based on symbolic evaluation of programs. Symbolic evaluation is an advanced static symbolic analysis that is centered around symbolic variable values, assumptions ab ...

3

Bell: bit-encoding online memory leak detection
Michael D. Bond, Kathryn S. McKinley
October 2006 **ACM SIGPLAN Notices , ACM SIGOPS Operating Systems Review , ACM SIGARCH Computer Architecture News , Proceedings of the 12th international conference on Architectural support for programming languages and operating systems ASPLOS-XII**, Volume 41 , 40 , 34 Issue 11 , 5 , 5
Publisher: ACM Press
Full text available: pdf(445.91 KB) Additional Information: full citation, abstract, references, index terms

Memory leaks compromise availability and security by crippling performance and crashing programs. Leaks are difficult to diagnose because they have no immediate symptoms. Online leak detection tools benefit from storing and reporting per-object *sites* (e.g., allocation sites) for potentially leaking objects. In programs with many small objects, per-object sites add high space overhead, limiting their use in production environments.This paper introduces *Bit-Encoding Leak Location* (Be ...

Keywords: low-overhead monitoring, managed languages, memory leaks, probabilistic approaches

4 Cork: dynamic memory leak detection for garbage-collected languages
Maria Jump, Kathryn S. McKinley
January 2007 **ACM SIGPLAN Notices , Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages POPL '07**, Volume 42 Issue 1
Publisher: ACM Press
Full text available: pdf(651.25 KB) Additional Information: full citation, abstract, references, index terms

A *memory leak* in a garbage-collected program occurs when the program inadvertently maintains references to objects that it no longer needs. Memory leaks cause systematic heap growth, degrading performance and resulting in program crashes after perhaps days or weeks of execution. Prior approaches for detecting memory leaks rely on heap differencing or detailed object statistics which store state proportional to the number of objects in the heap. These overheads preclude their use on the sa ...

Keywords: dynamic, garbage collection, memory leak detection, memory leaks, runtime analysis

5 Context- and path-sensitive memory leak detection
Yichen Xie, Alex Aiken
September 2005 **ACM SIGSOFT Software Engineering Notes , Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering ESEC/FSE-13**, Volume 30 Issue 5
Publisher: ACM Press
Full text available: pdf(251.65 KB) Additional Information: full citation, abstract, references, citings, index terms

We present a context- and path-sensitive algorithm for detecting memory leaks in programs with explicit memory management. Our leak detection algorithm is based on an underlying escape analysis: any allocated location in a procedure *P* that is not deallocated in *P* and does not escape from *P* is leaked. We achieve very precise context- and path-sensitivity by expressing our analysis using boolean constraints. In experiments with six large open source projects our analysis produ ...

Keywords: boolean satisfiability, error detection, memory leaks, memory management,

program analysis

6 Memory leak detection in C++

Cal Erickson

June 2003 **Linux Journal**, Volume 2003 Issue 110

Publisher: Specialized Systems Consultants, Inc.

Full text available: html(15.05 KB) Additional Information: full citation, abstract, index terms

It's never too soon to fix bugs, and you canstart using these tools as soon as your project will compile.

7 Memory allocation: Gated memory control for memory monitoring, leak detection and garbage collection

Chen Ding, Chengliang Zhang, Xipeng Shen, Mitsunori Ogihara

June 2005 **Proceedings of the 2005 workshop on Memory system performance MSP '05**

Publisher: ACM Press

Full text available: pdf(135.93 KB) Additional Information: full citation, abstract, references, citings, index terms

In the past, program monitoring often operates at the code level, performing checks at function and loop boundaries. Recent research shows that profiling analysis can identify high-level phases in complex binary code. Examples are time steps in scientific simulations and service cycles in utility programs. Because of their larger size and more predictable behavior, program phases make it possible for more accurate and longer term predictions of program behavior, especially its memory usage. This ...

Keywords: memory leak, memory usage monitoring, object life, preventive memory management, program phase

8 Research papers: test & analysis III: Static detection of leaks in polymorphic containers

David L. Heine, Monica S. Lam

May 2006 **Proceeding of the 28th international conference on Software engineering ICSE '06**

Publisher: ACM Press

Full text available: pdf(219.64 KB) Additional Information: full citation, abstract, references, index terms

This paper presents the first practical static analysis tool that can find memory leaks and double deletions of objects held in polymorphic containers. This is especially important since most dynamically allocated objects are stored in containers.The tool is based on the concept of object ownership: every object has one and only one *owning* pointer. The owning pointer holds the exclusive right and obligation to either delete the object or to transfer the obligation. This paper presents a n ...

Keywords: error detection, memory leaks, memory management, program analysis, type systems

9 Memory leak detection in embedded systems

Cal Erickson

September 2002 **Linux Journal**, Volume 2002 Issue 101

Publisher: Specialized Systems Consultants, Inc.

Full text available: html(13.11 KB) Additional Information: full citation, abstract, index terms

Erickson discusses some of the best tools for memory leak detectionfor embedded programmers.

10 On the usefulness of type and liveness accuracy for garbage collection and leak detection

Martin Hirzel, Amer Diwan, Johannes Henkel

November 2002 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 24 Issue 6

Publisher: ACM Press

Full text available: pdf(684.85 KB) Additional Information: full citation, abstract, references, citings, index terms

The effectiveness of garbage collectors and leak detectors in identifying dead objects depends on the *accuracy* of their reachability traversal. Accuracy has two orthogonal dimensions: (i) whether the reachability traversal can distinguish between pointers and nonpointers (*type accuracy*), and (ii) whether the reachability traversal can identify memory locations that will be dereferenced in the future (*liveness accuracy*). This article presents an experimental study of the impo ...

Keywords: Conservative garbage collection, leak detection, liveness accuracy, program analysis, type accuracy

11 A practical flow-sensitive and context-sensitive C and C++ memory leak detector

David L. Heine, Monica S. Lam

May 2003 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation PLDI '03**, Volume 38 Issue 5

Publisher: ACM Press

Full text available: pdf(214.44 KB) Additional Information: full citation, abstract, references, citings, index terms

This paper presents a static analysis tool that can automatically find memory leaks and deletions of dangling pointers in large C and C++ applications.We have developed a type system to formalize a practical ownership model of memory management. In this model, every object is pointed to by one and only one *owning* pointer, which holds the exclusive right and obligation to either delete the object or to transfer the right to another owning pointer. In addition, a pointer-typed class member ...

Keywords: error detection, memory leaks, memory management, program analysis, type systems

12 Static detection of dynamic memory errors

David Evans

May 1996 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1996 conference on Programming language design and implementation PLDI '96**, Volume 31 Issue 5

Publisher: ACM Press

Full text available: pdf(1.17 MB) Additional Information: full citation, abstract, references, citings, index terms

Many important classes of bugs result from invalid assumptions about the results of functions and the values of parameters and global variables. Using traditional methods, these bugs cannot be detected efficiently at compile-time, since detailed cross-procedural analyses would be required to determine the relevant assumptions. In this work, we introduce annotations to make certain assumptions explicit at interface points. An efficient static checking tool that exploits these annotations can dete ...

13 HeapMD: identifying heap-based bugs using anomaly detection

Trishul M. Chilimbi, Vinod Ganapathy

October 2006 **ACM SIGARCH Computer Architecture News , ACM SIGOPS Operating Systems Review , ACM SIGPLAN Notices , Proceedings of the 12th international conference on Architectural support for programming languages and operating systems ASPLOS-XII**, Volume 34 , 40 , 41 Issue 5 , 5 , 11

Publisher: ACM Press

Full text available: pdf(246.89 KB) Additional Information: full citation, abstract, references, index terms

We present the design, implementation, and evaluation of HeapMD, a dynamic analysis tool that finds heap-based bugs using anomaly detection. HeapMD is based upon the observation that, in spite of the evolving nature of the heap, several of its properties remain stable. HeapMD uses this observation in a novel way: periodically, during the execution of the program, it computes a suite of metrics which are sensitive to the state of the heap. These metrics track heap behavior, and the stability of t ...

Keywords: anomaly detection, bugs, debugging, heap, metrics

14 Efficient detection of all pointer and array access errors

Todd M. Austin, Scott E. Breach, Gurindar S. Sohi

June 1994 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1994 conference on Programming language design and implementation PLDI '94**, Volume 29 Issue 6

Publisher: ACM Press

Full text available: pdf(1.62 MB) Additional Information: full citation, abstract, references, citings, index terms

We present a pointer and array access checking technique that provides complete error coverage through a simple set of program transformations. Our technique, based on an extended safe pointer representation, has a number of novel aspects. Foremost, it is the first technique that detects all spatial and temporal access errors. Its use is not limited by the expressiveness of the language; that is, it can be applied successfully to compiled or interpreted languages with subscripted and mutabl ...

15 AccMon: Automatically Detecting Memory-Related Bugs via Program Counter-Based Invariants

Pin Zhou, Wei Liu, Long Fei, Shan Lu, Feng Qin, Yuanyuan Zhou, Samuel Midkiff, Josep Torrellas

December 2004 **Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture MICRO 37**

Publisher: IEEE Computer Society

Full text available: pdf(249.11 KB) Additional Information: full citation, abstract, citings

This paper makes two contributions to architectural support for software debugging. First, it proposes a novel statistics-based, on-the-fly bug detectionmethod called PC-based invariant detection. The idea is based on the observation that, in most programs, a given memory location is typically accessed by only a few instructions. Therefore, by capturing the invariant of the set of PCs that normally access a given variable, we can detect accesses by outlier instructions, which are often caused by ...

16 OOPSLA practitioner reports chair's welcome: The dynamics of changing dynamic memory allocation in a large-scale C++ application

Neil B. Harrison, John H. Meiners

October 2006 **Companion to the 21st ACM SIGPLAN conference on Object-oriented**

programming systems, languages, and applications OOPSLA '06

Publisher: ACM Press

Full text available: pdf(229.94 KB) Additional Information: full citation, abstract, references, index terms

Changing the approach to dynamic memory allocation in a large legacy application is challenging. In order to improve the robustness of memory allocation, we fundamentally changed it. We replaced standard heap allocation with class-specific heaps. We were able to do it with almost no changes to the existing class code by overriding the C++ new() and delete() operators, and using templates creatively to insert the changes into class hierarchies.The results have been very positive. Misuse of dynami ...

Keywords: curiously recurring template pattern, memory management, templates

17 ARCHER: using symbolic, path-sensitive analysis to detect memory access errors

Yichen Xie, Andy Chou, Dawson Engler

September 2003 **ACM SIGSOFT Software Engineering Notes , Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering ESEC/FSE-11**, Volume 28 Issue 5

Publisher: ACM Press

Full text available: pdf(582.35 KB) Additional Information: full citation, abstract, references, citings, index terms

Memory corruption errors lead to non-deterministic, elusive crashes. This paper describes ARCHER (*ARray CHeckER*) a static, effective memory access checker. ARCHER uses path-sensitive, interprocedural symbolic analysis to bound the values of both variables and memory sizes. It evaluates known values using a constraint solver at every array access, pointer dereference, or call to a function that expects a size parameter. Accesses that violate constraints are flagged as errors. Those that ar ...

Keywords: buffer overflow, buffer overrun, error detection, memory access errors, security, static analysis

18 Understanding memory allocation of scheme programs

Manuel Serrano, Hans-J. Boehm

September 2000 **ACM SIGPLAN Notices , Proceedings of the fifth ACM SIGPLAN international conference on Functional programming ICFP '00**, Volume 35 Issue 9

Publisher: ACM Press

Full text available: pdf(821.49 KB) Additional Information: full citation, abstract, references, citings, index terms

Memory is the performance bottleneck of modern architectures. Keeping memory consumption as low as possible enables fast and unobtrusive applications. But it is not easy to estimate the memory use of programs implemented in functional languages, due to both the complex translations of some high level constructs, and the use of automatic memory managers.To help understand memory allocation behavior of Scheme programs, we have designed two complementary tools. The first one reports on frequency of ...

19 LIFT: A Low-Overhead Practical Information Flow Tracking System for Detecting Security Attacks

Feng Qin, Cheng Wang, Zhenmin Li, Ho-seop Kim, Yuanyuan Zhou, Youfeng Wu

December 2006 **Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture MICRO '06**

Publisher: IEEE Computer Society

Full text available: Additional Information:

pdf(254.69 KB) full citation, abstract

Computer security is severely threatened by software vulnerabilities. Prior work shows that information flow tracking (also referred to as taint analysis) is a promising technique to detect a wide range of security attacks. However, current information flow tracking systems are not very practical, because they either require program annotations, source code, non-trivial hardware extensions, or incur prohibitive runtime overheads. This paper proposes a low overhead, software-only information flow ...

20 Checkmate: cornering C++ dynamic memory errors with checked pointers

Scott M. Pike, Bruce W. Weide, Joseph E. Hollingsworth

March 2000 **ACM SIGCSE Bulletin , Proceedings of the thirty-first SIGCSE technical symposium on Computer science education SIGCSE '00**, Volume 32 Issue 1

Publisher: ACM Press

Full text available: pdf(554.01 KB) Additional Information: full citation, abstract, references, citings, index terms

Pointer errors are stumbling blocks for student and veteran programmers alike. Although languages such as Java use references to protect programmers from pointer pitfalls, the use of garbage collection dictates that languages like C++ will still be used for real-time mission-critical applications. Pointers will stay in the classroom as long as they're used in industry, so as educators, we must find better ways to teach them. This paper presents checked pointers, a simple wr ...

Results 1 - 20 of 200 Result page: **1** 2 3 4 5 6 7 8 9 10 next